Tutorial on Keras

CAP 6412 - ADVANCED COMPUTER VISION

SPRING 2018

KISHAN S ATHREY

Deep learning packages

- **TensorFlow** – Google
- **PyTorch** – Facebook AI research
- **Keras** – Francois Chollet (now at Google)
- **Chainer** – Company in Japan
- **Caffe** - Berkeley Vision and Learning Center
- **CNTK** - Microsoft

https://www.slideshare.net/0xdata/deep-learning-with-mxnet-dmitry-larko

Overview of the tutorial

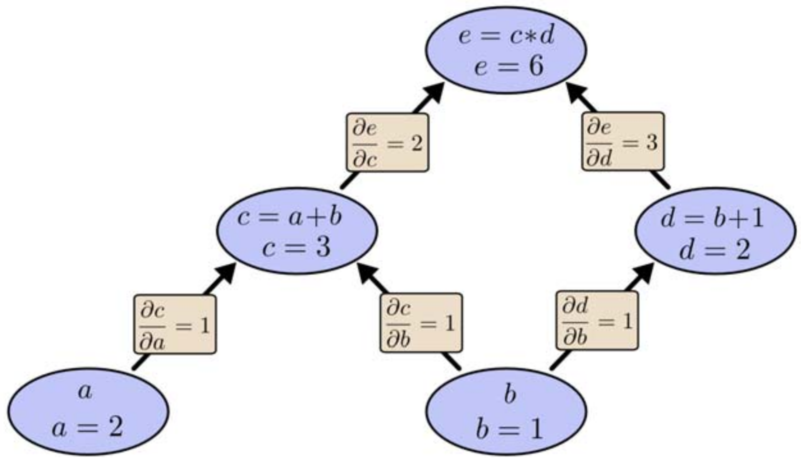
- What is Keras ?
- Basics of Keras environment
- Building Convolutional neural networks
- Building Recurrent neural networks
- Introduction to other types of layers
- Introduction to Loss functions and Optimizers in Keras
- Using Pre-trained models in Keras
- Saving and loading weights and models
- Popular architectures in Deep Learning

What is Keras ?

- **Deep neural network library in Python**
  - High-level neural networks API
  - Modular – Building model is just stacking layers and connecting computational graphs
  - Runs on top of either TensorFlow or Theano or CNTK
- **Why use Keras ?**
  - Useful for fast prototyping, ignoring the details of implementing backprop or writing optimization procedure
  - Supports Convolution, Recurrent layer and combination of both.
  - Runs seamlessly on CPU and GPU
  - Almost any architecture can be designed using this framework
  - Open Source code – Large community support

Working principle - Backend

- **Computational Graphs**
  - Expressing complex expressions as a combination of simple operations
  - Useful for calculating derivatives during backpropagation
  - Easier to implement distributed computation
  - Just specify the inputs, outputs and make sure the graph is connected



*e = c*d*
where, *"c = a+b"* and *"d = b+1"*
So, *e = (a+b)*(b+1)*
Here *"a"*, *"b"* are inputs

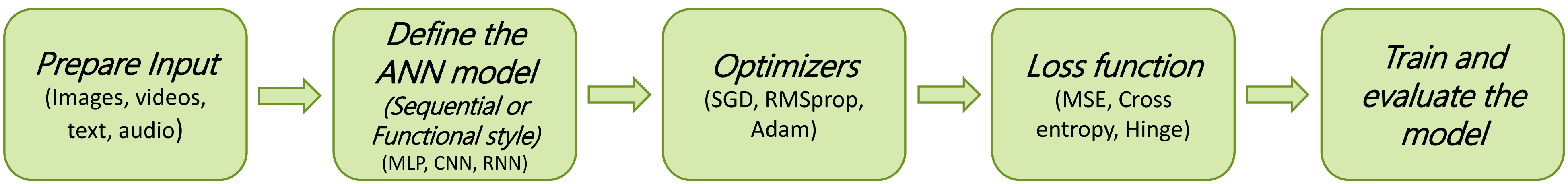
General pipeline for implementing an ANN

- Design and define the neural network architecture
- Select the optimizer that performs optimization (gradient descent)
- Select the loss function and train it
- Select the appropriate evaluation metric for the given problem

Implementing a neural network in Keras

- **Five major steps**
  - Preparing the input and specify the input dimension (size)
  - Define the model architecture and build the computational graph
  - Specify the optimizer and configure the learning process
  - Specify the Inputs, Outputs of the computational graph (model) and the Loss function
  - Train and test the model on the dataset

Note: Gradient calculations are taken care by Auto – Differentiation and parameter updates are done automatically in the backend



Procedure to implement an ANN in Keras

- Importing *Sequential class* from keras.models

```
from keras.models import Sequential

model = Sequential()
```

- Stacking layers using .add() method

```
model.add(Dense(units=64, input_dim=100))
model.add(Activation('relu'))
model.add(Dense(units=10))
model.add(Activation('softmax'))
```

- Configure learning process using .compile() method

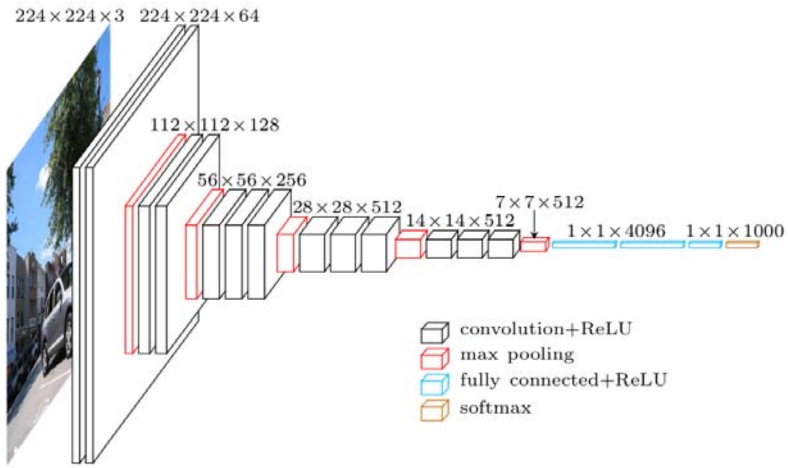
```
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

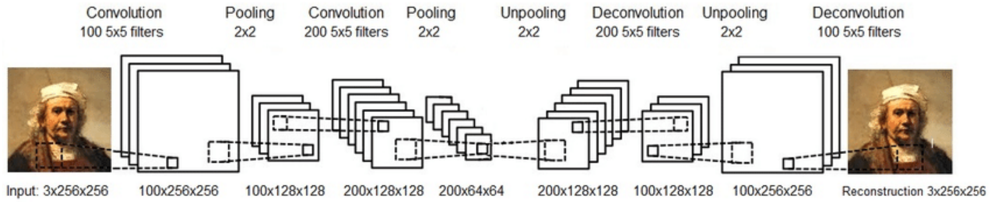
- Train the model on train dataset using .fit() method

```
model.fit(x_train, y_train, epochs=5, batch_size=32)
```

Keras models – Sequential

- Sequential model
- Linear stack of layers
- Useful for building simple models
  - Simple classification network
  - Encoder – Decoder models

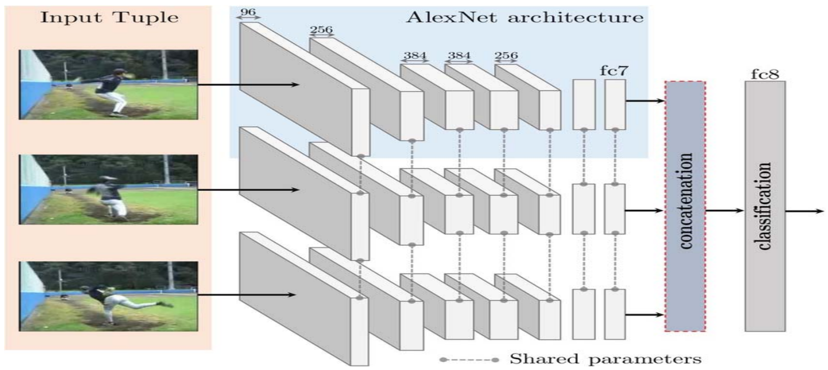




[1] https://blog.heuritech.com/2016/02/29/a-brief-report-of-the-heuritech-deep-learning-meetup-5/vgg16/
[2] https://www.cc.gatech.edu/~hays/7476/projects/Avery_Wenchen/

Keras models – Functional

- Functional Model
  - Multi – input and Multi – output models
  - Complex models which forks into 2 or more branches
  - Models with shared (Weights) layers



[1] https://www.sciencedirect.com/science/article/pii/S0263224117304517
[2] Unsupervised Domain Adaptation by Backpropagation, https://arxiv.org/abs/1409.7495

Keras models – Functional (Domain Adaption)

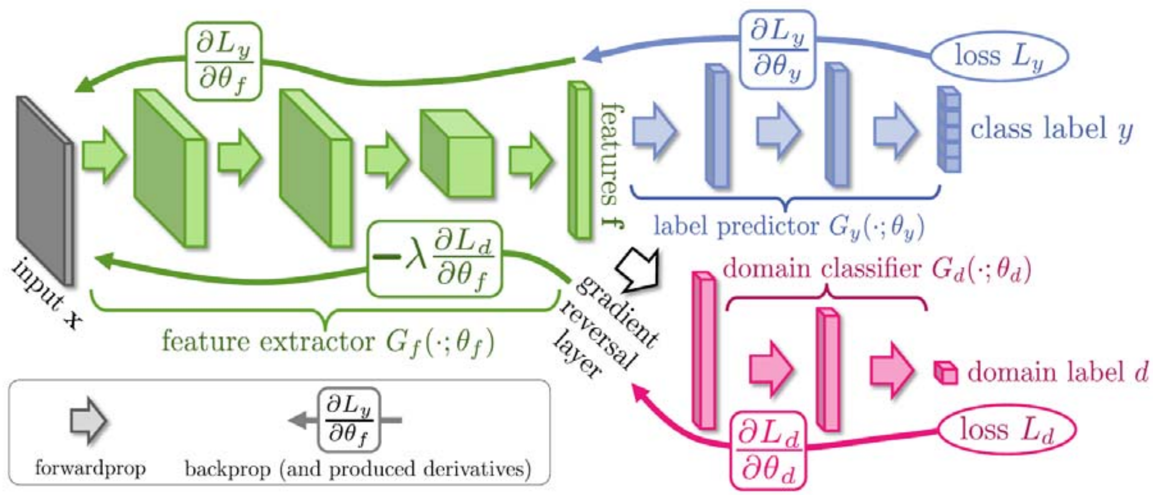
(a) MNIST

Domain A
With Labels

(b) SVHN

Domain B
Without Labels

- Train on Domain A and Test on Domain B
- Results in poor performance on test set
- The data are from different domains
- **Solution:** Adapt the model to both the domains



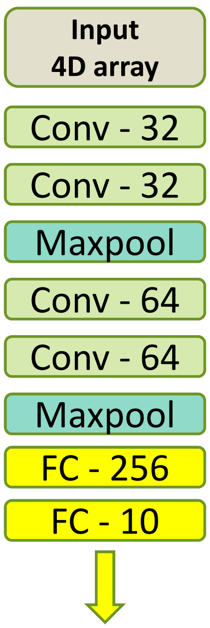
[1] https://www.sciencedirect.com/science/article/pii/S0263224117304517
[2] Unsupervised Domain Adaptation by Backpropagation, https://arxiv.org/abs/1409.7495

Convolution neural network - Sequential model

- Mini VGG style network
- FC – Fully Connected layers (dense layer)
- Input dimension – 4D
  - [N_Train, height, width, channels]
  - **N_train** – Number of train samples

- **Height** – height of the image
- **Width** – Width of the image
- **channels** – Number of channels
- For RGB image, **channels** = 3
- For gray scale image, **channels** = 1


```
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.optimizers import SGD

# Generate dummy data
x_train = np.random.random((100, 100, 100, 3))
y_train = keras.utils.to_categorical(np.random.randint(10, size=(100, 1)), num_classes=10)
x_test = np.random.random((20, 100, 100, 3))
y_test = keras.utils.to_categorical(np.random.randint(10, size=(20, 1)), num_classes=10)
```




```
model = Sequential()
# input: 100x100 images with 3 channels -> (100, 100, 3) tensors.
# this applies 32 convolution filters of size 3x3 each.
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd)

model.fit(x_train, y_train, batch_size=32, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=32)
```

Simple MLP network - Functional model

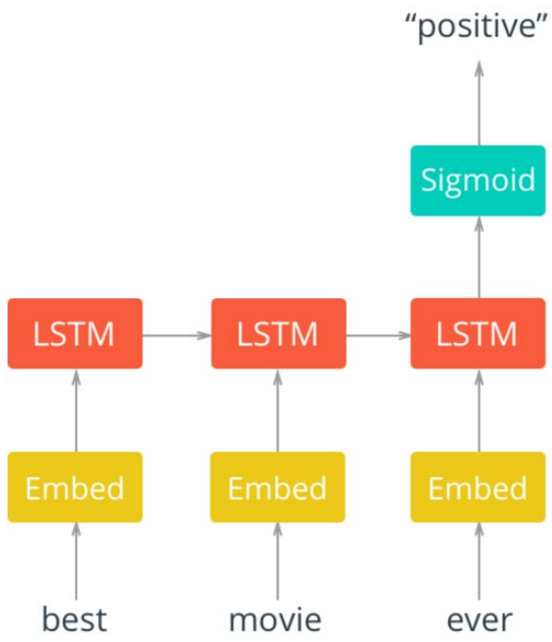
- Import class called "Model"
- Each layer explicitly returns a tensor
- Pass the returned tensor to the next layer as input
- Explicitly mention model inputs and outputs

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
```

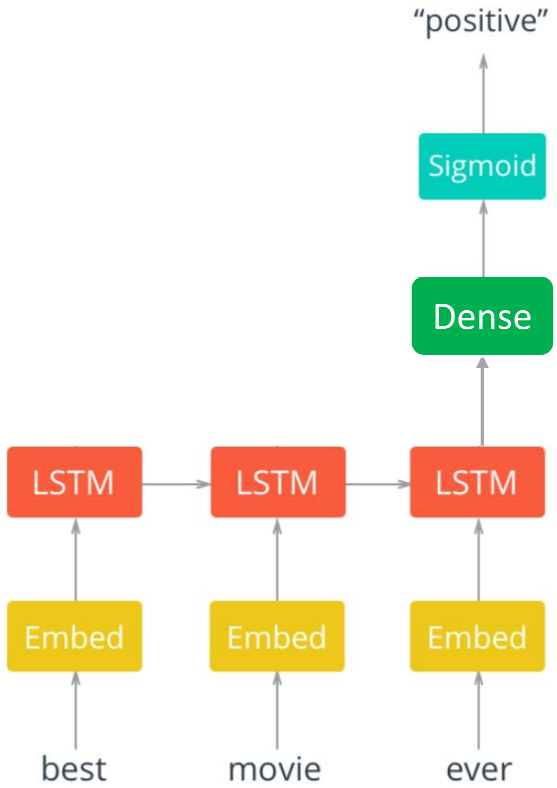
```
# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

Recurrent Neural Networks

- RNNs are used on sequential data – Text, Audio, Genomes etc.
- Recurrent networks are of three types
  - Vanilla RNN
  - LSTM
  - GRU
- They are feedforward networks with internal feedback
- The output at time "t" is dependent on current input and previous values



Recurrent Neural Network




```
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.layers import Embedding
from keras.layers import LSTM

model = Sequential()
model.add(Embedding(max_features, output_dim=256))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=16, epochs=10)
score = model.evaluate(x_test, y_test, batch_size=16)
```

Convolution layers

- 1D Conv

  keras.layers.convolutional.Conv1D(filters, kernel_size, strides=1, padding='valid', dilation_rate=1, activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

  Applications: Audio signal processing, Natural language processing

- 2D Conv

  keras.layers.convolutional.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format=**None**, dilation_rate=(1, 1), activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

  Applications: Computer vision - **Images**

- 3D Conv

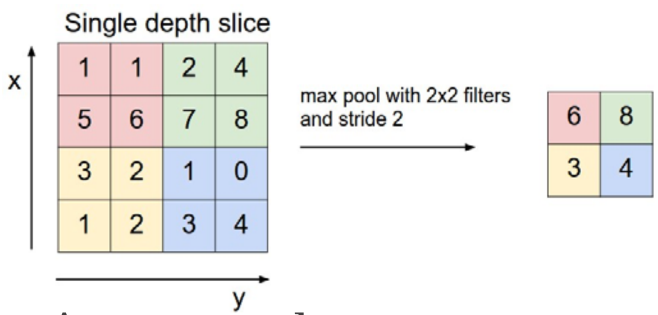
  keras.layers.convolutional.Conv3D(filters, kernel_size, strides=(1, 1, 1), padding='valid', data_format=**None**, dilation_rate=(1, 1, 1), activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

  Applications: Computer vision – **Videos** (Convolution along temporal dimension)

Pooling layers

- Max pool

keras.layers.pooling.MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')



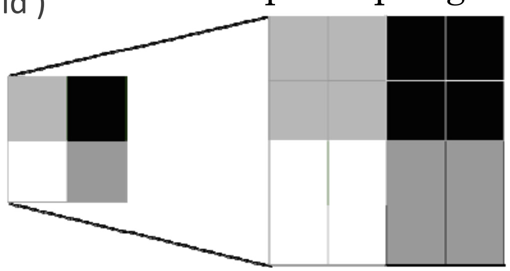
- Average pool

keras.layers.pooling.AveragePooling2D(pool_size=(2, 2), strides=None, padding='valid')

- Up sampling

keras.layers.convolutional.UpSampling2D(size=(2, 2))

Up sampling



General layers

- Dense

keras.layers.core.Dense(units, activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

- Dropout

keras.layers.core.Dropout(rate, noise_shape=**None**, seed=**None**)

- Embedding

keras.layers.embeddings.Embedding(input_dim, output_dim, input_length=**None** embeddings_initializer='uniform', embeddings_regularizer=**None**, activity_regularizer=**None**, embeddings_constraint=**None**, mask_zero=**False**)

Optimizers available in Keras

- How do we find the “best set of parameters (weights and biases)” for the given network ?
- **Optimization**
  - They vary in the speed of convergence, ability to avoid getting stuck in local minima
  - SGD – Stochastic gradient descent
  - SGD with momentum
  - Adam
  - AdaGrad
  - RMSprop
  - AdaDelta
- Detailed explanation of each optimizer is given in the “Deep learning book”
  - URL: http://www.deeplearningbook.org/contents/optimization.html

Loss functions available in Keras

- **MSE – Mean square error**

$$\text{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$$

- **MAE – Mean absolute error**

$$\text{MAE} = \frac{1}{n}\sum_{t=1}^{n} |e_t|$$

- **Categorical cross entropy** – "K" number of classes

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log P(y^{(i)} = k|x^{(i)};\theta)\right]$$

- **KL divergence** – If P(X) and Q(X) are two different probability distributions, then we can measure how different these two distributions are using KL divergence

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{\mathrm{x}\sim P}\left[\log\frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x}\sim P}\left[\log P(x) - \log Q(x)\right]$$

Loading and Saving Keras models

- Use *.save* method to save the model
- Use *load_model* function to load saved model
- Saved file contains –
  - Architecture of the model
  - Weights and biases
  - State of the optimizer
- Saving weights
- Loading all the weights and loading weights layer wise

```
from keras.models import load_model

model.save('my_model.h5')  # creates a HDF5 file 'my_model.h5'
del model  # deletes the existing model

# returns a compiled model
# identical to the previous one
model = load_model('my_model.h5')
```

```
model.save_weights('my_model_weights.h5')
```

```
model.load_weights('my_model_weights.h5', by_name=True)
```

Extracting features from pre-trained models

- Import the network [eg:VGG16]
- Specify the weights
- Specify whether the classifier at the top has to be included or not
- The argument *"include_top = False"* – removes the classifier from the imported model
- The input size of the image must be same as what the imported model was trained on (with exceptions)

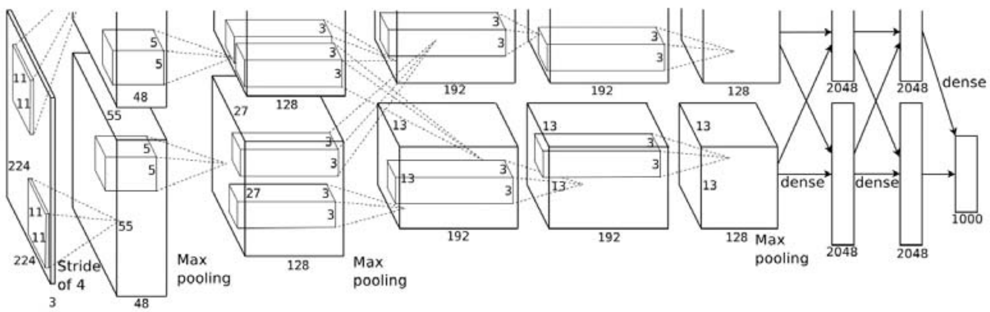
```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

model = VGG16(weights='imagenet', include_top=False)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

Popular Deep learning Architectures

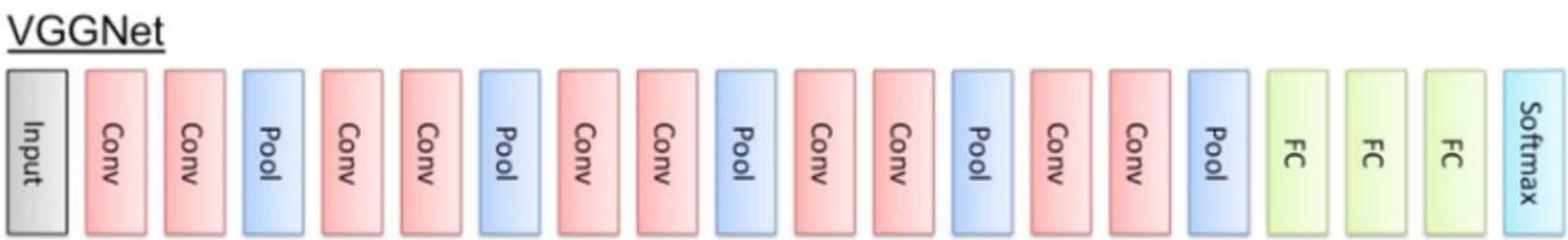
- Popular Convolution networks
  - Alex net
  - VGG
  - Res-Net
  - DenseNet
- Generative models
  - Autoencoders
  - Generative adversarial networks

Image recognition networks

- AlexNet – 2012



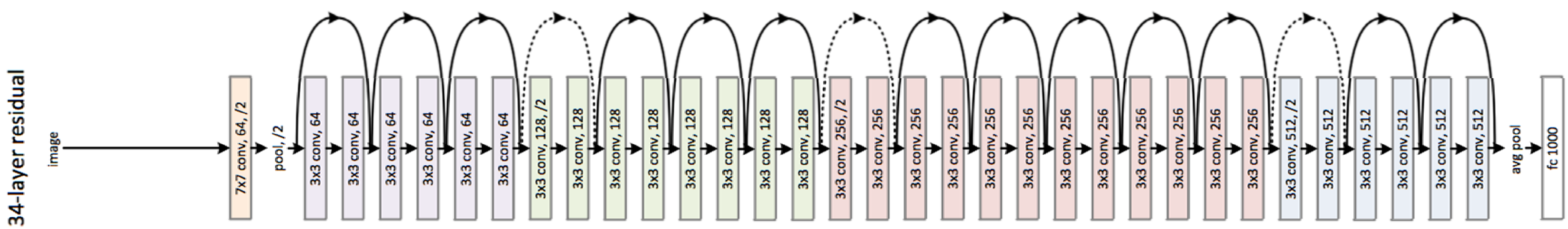
- VGG - 2014



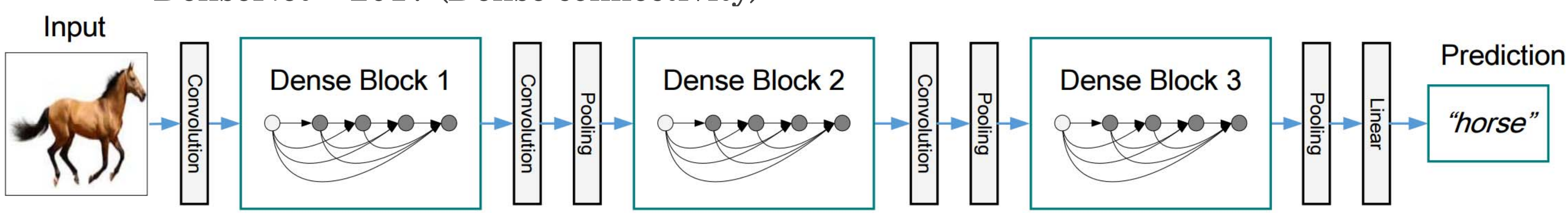
[1] AlexNet, https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf
[2] VGG Net, https://arxiv.org/pdf/1409.1556.pdf

Image recognition networks

- ResNet – 2015 (residual connections)

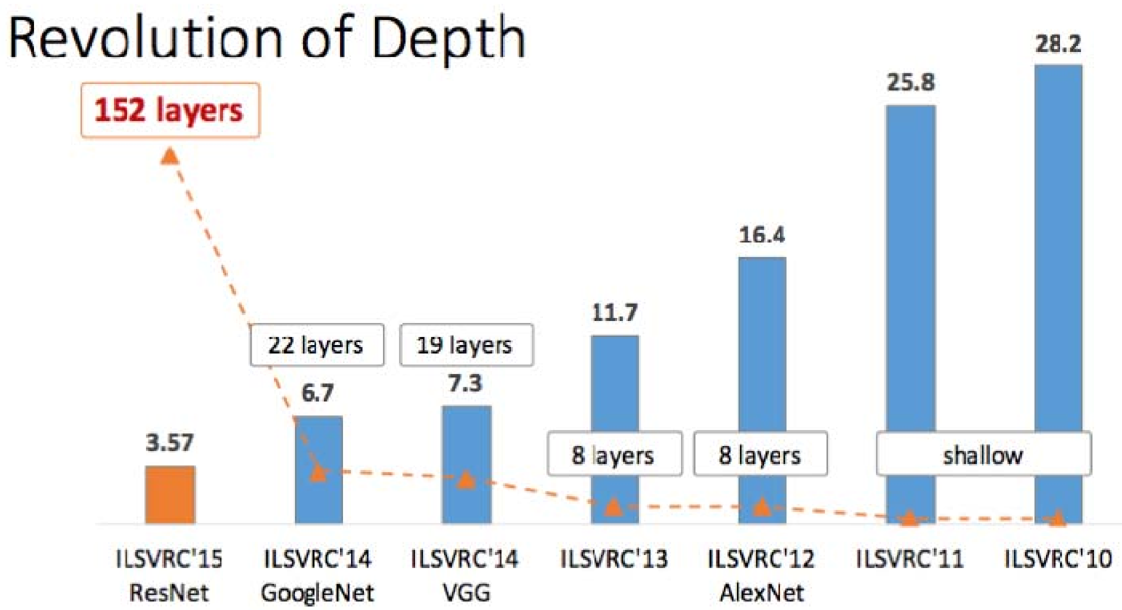


- DenseNet – 2017 (Dense connectivity)

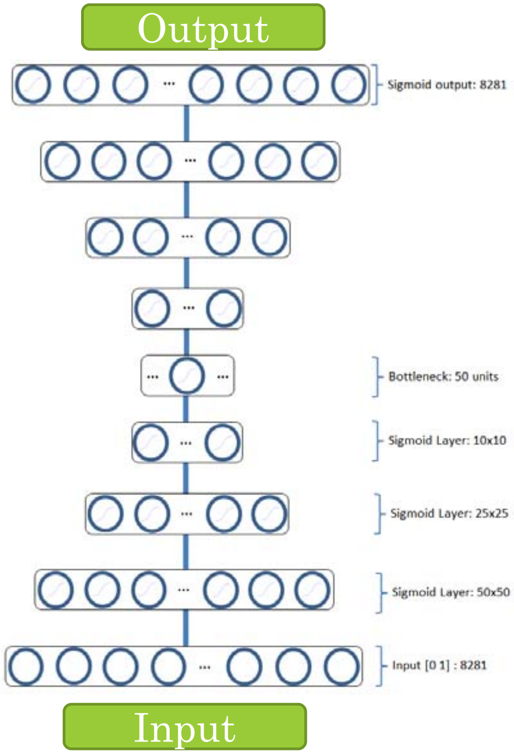


[1] ResNet, https://arxiv.org/abs/1512.03385
[2] DenseNet, https://arxiv.org/abs/1608.06993

Performance of the recognition networks



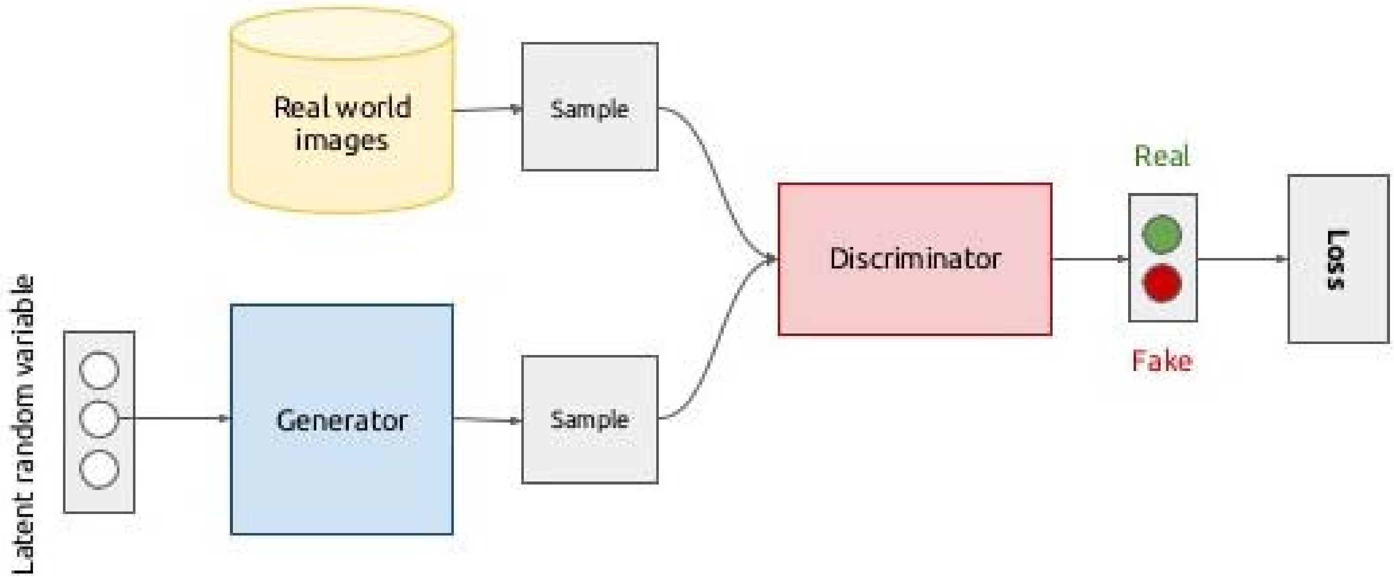
Autoencoders



- Unsupervised representation learning
- Dimensionality reduction
- Denoising

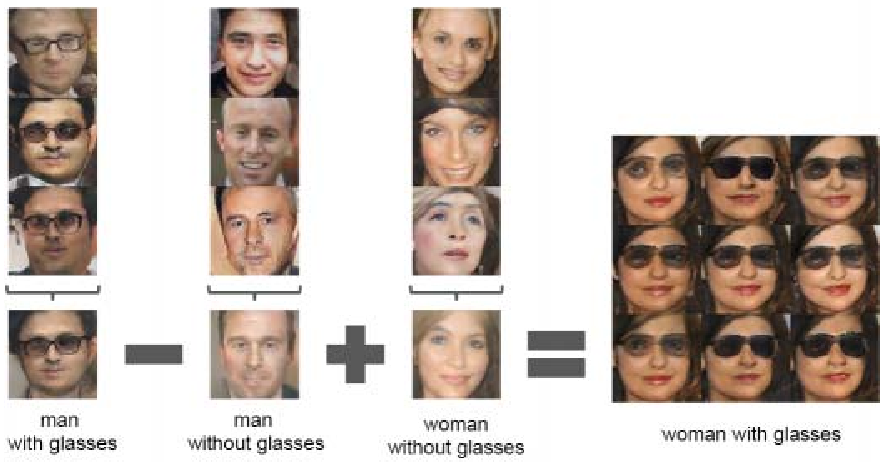
https://www.researchgate.net/figure/Figure-9-A-autoencoder-with-many-hidden-layers-two-stacked-autoencoders_282997080_fig9

Generative Adversarial Network



Interesting Applications using GANs

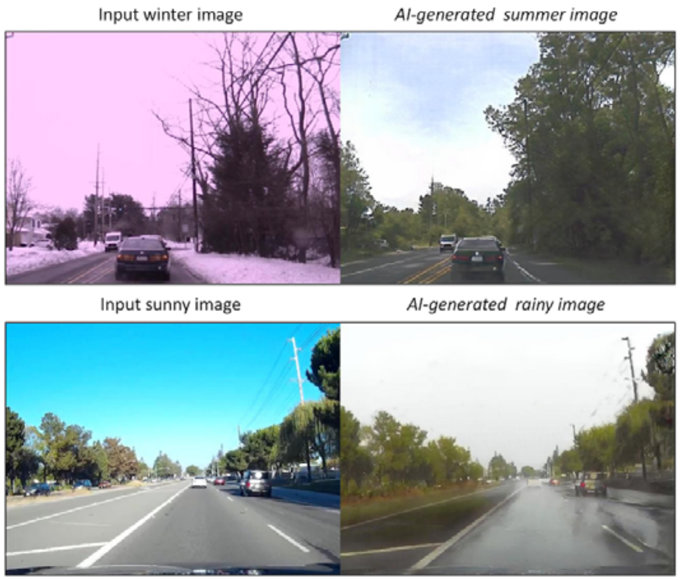
- Generate images from textual description
- Performing arithmetic in latent space

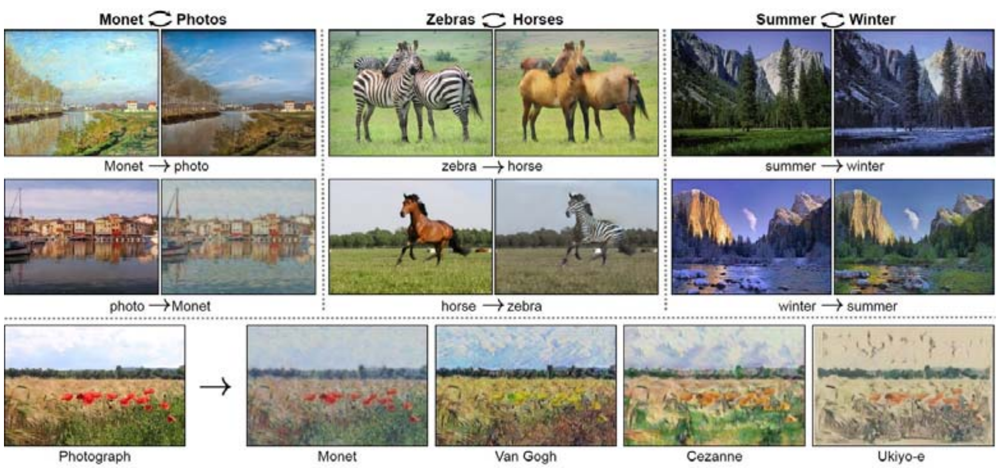


[1] Stack GAN, https://arxiv.org/abs/1612.03242
[2] DC GAN, https://arxiv.org/abs/1511.06434

Interesting Applications using GANs

- Generate images of the same scene with different weather conditions
- Transfer the style of painting from one image to other
- Change the content in the image





[1] UNIT, https://arxiv.org/pdf/1703.00848
[2] Cyclic GAN, https://arxiv.org/abs/1703.10593

Community contributed layers and other functionalities

https://github.com/farizrahman4u/keras-contrib/tree/master/keras_contrib

https://github.com/fchollet/keras/tree/master/keras/layers

Keras Documentation – keras.io

Keras Blog - https://blog.keras.io/index.html

Questions ?